CHAPTER 3

SOLVING SIMPLE PROBLEMS WITH BASIC

END, PRINT, LET, Constants, Variables, and Arithmetic Operations

WRITING SIMPLE BASIC PROGRAMS

Learning to solve simple problems in BASIC is somewhat like learning to write and communicate in a foreign language, with one exception—BASIC is much easier to learn. By comparison, in the average pocket dictionary there are approximately 50,000 words, but there are less than 50 keywords in BASIC.

You could probably learn all these keywords in one day; however, it takes longer than one day to become a proficient programmer in the BASIC language. As with any language, knowing only the grammar or mechanics of the language is not sufficient. You also must know how to use the language to instruct the computer to solve problems.

As discussed in Chapter 1, the first step in writing a program is problem definition and understanding. Whether it is an elaborate application or just a simple "one time" program you wish to write, you first must understand the problem, then decide how you are going to solve it. Once this is done, you start writing the program. Challenging as it is, this can be fun; although at times it can be very frustrating. As you progress through the course, make note of how each keyword is used to instruct the computer to produce the desired result;

In this chapter, we will discuss three statements that can be used in writing simple BASIC programs and give examples of programs. The three keywords used to construct these statements are END, PRINT, and LET.

END Statement

Every program requires an **END** statement, so we'll start with it and its functions. The END statement has two functions—to indicate to the compiler (interpreter) that there are no more BASIC statements for it to translate and to terminate execution of the program. Execution of an END statement causes the computer to print a message which indicates program execution is terminated and that the computer is ready for further processing.

The format of the END statement is:

999 END

statement number keyword

Remember, the last statement in any BASIC program must be the single keyword END. This means that the statement number must be the largest statement number in the program. There is no additional information required or allowed in the END statement.

PRINT Statement

The next area of consideration is printing and the uses of the **PRINT** statement. BASIC is different from other high level programming languages in that it has a predefine format for printed output. This format, referred to as *standard* spacing, divides the print line into a specified number of print zones or fields of a predefine length. The number and length of print zones may vary with different computers. For the purpose of this text, we will use 16 spaces per print field. Additionally, *packed* spacing maybe achieved through the use of punctuation in the PRINT statement.

The function of the PRINT statement is to instruct the computer to output something, either on the terminal or the printer. It can be used in several different ways. One way is to print the value of a single variable.

Example:

Printing the value of a single variable

```
30 PRINT A
99 END
RUN
```

This statement will cause the computer to print whatever value it has for the variable A. The output will be the value of A only, the variable name A will not be printed. Let's assume the value of A in the PRINT statement is 896. When the PRINT statement is executed, the output will be:

```
896
```

Note that only the value of variable A was printed, not the variable name.

Now let's suppose you wanted to print the values of several variables. It could be done this way:

Example:

Printing the values of three variables

```
10 PRINT A,B,C
99 END
RUN
```

Assume the values of A, B, and C are 896, 754, and 969 respectively. The output from the PRINT statement would look like this:

```
896          754          969
```

Whenever you wish to print several variables with one PRINT statement, you must separate them with either commas or semicolons, as was done in line number 10 of the example. The commas will give you standard spacing. If semicolons had been used, we would have gotten packed spacing. More on the use of the comma and semicolon will be discussed later in the chapter.

Printing the results of a computation is another use of the PRINT statement. Let's suppose we want to get the average of three scores: 82, 95, and 93. First we must define the problem: how to compute an average. You add up the scores and divide by the number of scores. This problem can be solved by using a PRINT statement like this:

Example:

Print the results of a computation

```
15  PRINT (82+95+93)/3
99  END
RUN
```

We have used a simple mathematical formula in conjunction with the PRINT statement to solve the problem. First the computer will add all the scores, then it will divide by three and print the result. When this statement is executed, the output will be:

```
 90
```

This output is rather plain and meaningless to anyone other than the person who wrote the program. There is another use of the PRINT statement which will enhance the output from the previous example, printing a message using the PRINT statement. Any message to be printed must be enclosed in quotation marks.

Example:

Printing a message

```
10  PRINT "THE AVERAGE SCORE IS"
20  PRINT (82+95+93)/3
30  END
RUN
```

The output from these PRINT statements would be:

```
THE AVERAGE SCORE IS
 90
```

You will notice the quotation marks around the message in line 10 causes the message to be printed just as it appears in the program. Using the PRINT statement in this manner adds clarity and meaning to the output.

Now let's look at another combination of uses of the PRINT statement. Suppose we wanted to print out the individual scores and the average of the scores, with a blank line between them. The following example shows how ' this could be done:

Example:

Printing blank lines

```
10 PRINT "INDIVIDUAL SCORES"
20 PRINT 82
30 PRINT 95
40 PRINT 93
50 PRINT
60 PRINT "THE AVERAGE SCORE IS"
70 PRINT (82+95+93)/3
80 END
RUN
```

Note that line 50 contains a blank PRINT statement. This will cause spacing between lines of output. It can be used to provide clarity and improve readability. When the program is executed, the output will look like this:

```
INDIVIDUAL SCORES
 82
 95
 93

THE AVERAGE SCORE IS
 90
```

As seen in this example the blank PRINT statement produced a blank line between the individual scores and the average score. This feature may work differently on some computers.

We could have written the previous example in a different manner—using semicolons to separate messages and numbers.

Example:

Printing messages and numbers on the same line,

```
10   PRINT "INDIVIDUAL SCORES ARE ";82;95;93
20   PRINT
30   PRINT "THE AVERAGE SCORE IS ";(82+95+93)/3
40   END
RUN
```

The information enclosed in quotation marks will be printed exactly as it appears in the program followed by the numeric data in line 10 and the result of the computation in line 30.

```
INDIVIDUAL SCORES ARE    82    95    93

THE AVERAGE SCORE IS    90
```

The various uses of the PRINT statement have been presented. Let's take a closer look at the punctuation used in PRINT statements, and how it affects the output. The two primary punctuation marks used in a PRINT statement are the comma and semicolon.

COMMA.—When a comma is used as a separator in a PRINT statement, standard spacing (16 spaces per field) is achieved. When numbers are printed, the first space is reserved for a minus sign. A comma maybe used at the end of a PRINT statement to allow the information in the following PRINT statement to be printed on the same line. On some computers, when two commas (,,) are used together they will produce a blank field in printed output.

Here are some examples of what commas do in a PRINT statement.

Example:

Using commas as separators

```
10   PRINT "INDIVIDUAL SCORES"
20   PRINT 82,95,93
30   END
RUN
```

The output from this example would look like this:

```
INDIVIDUAL SCORES
 82          95          93
```

You will notice the output is spaced out into three print zones or fields.

Look at the following example and see what effect a comma at the end of a PRINT statement has on the output.

Example:

Using commas at the end of PRINT statements

```
10  PRINT "THE AVERAGE SCORE IS",
20  PRINT (82+95+93)/3
30  END
RUN
```

The output will look like this:

```
THE AVERAGE SCORE IS    90
```

Here we have two PRINT statements but only one line of output. The comma at the end of line 10 caused the output from line 20 to be printed on the same line as the output from line 10 beginning in the next available print zone.

Let's see what will happen if we use two commas (,,) together in a PRINT statement. (NOTE: This may not work on some computers.)

Example:

Using two commas together

```
10  PRINT "INDIVIDUAL SCORES"
20  PRINT 82,,95,93
30  END
RUN
```

The output will look like this:

```
INDIVIDUAL SCORES
 82                        95          93
```

Here we have a blank field. The second data element started in the third print zone or field instead of the second, because of the two commas used together.

SEMICOLON.—NOW let's examine the second punctuation mark used in PRINT statements, the semicolon. When a semicolon is used in a PRINT statement, messages are printed together without any spaces (packed spacing). However, numeric data will be printed with two spaces between each number, one space is reserved for the minus sign. Like the comma, a semicolon used at the end of a PRINT statement causes the information in the next PRINT statement to be printed on the same line. The following examples show the effect a semicolon has on a PRINT statement.

Example:

Using semicolons as separators

```
10   PRINT "INDIVIDUAL SCORES"
20   PRINT 82;95;93
30   END
RUN
INDIVIDUAL SCORES
 82    95    93
```

As seen in this output, the semicolon used as a separator in a PRINT statement causes packed spacing.

Example:

Using semicolons at the end of PRINT statements

```
10   PRINT "THE AVER";
20   PRINT "AGE SCORE IS ";
30   PRINT (82+95+93)/3
40   END
RUN
THE AVERAGE SCORE IS    90
```

As seen in the output from this example, the semicolons caused all the printed output to come out on the same line with the message printed together followed by the numeric data.

Some important points to remember about the PRINT statement are:

- It will print any message enclosed in quotation marks.
- It will print any number or variable.
- It will calculate and print the result of an expression.
- A blank PRINT statement will cause a blank line in your output.
- When a comma is used as a separator, normal spacing (16 spaces per field) is achieved.
- When a comma is used at the end of a PRINT statement, the data in the next PRINT statement is continued on the same line beginning in the next print zone.
- When " " (quote, blank, quote) is used as a print field the computer will skip to the next print field. You are actually telling the computer to print a blank field. On some computers, two commas (,,) together will produce the same results.
- When a semicolon is used as a separator, packed spacing is achieved.
- When a semicolon is used at the end of a PRINT statement. the information contained in the next PRINT statement is continued on the same line.

PRINT statements can become awkward when many computations are to be done or the results of computations are to be used in other calculations. *The results of computations done in a PRINT statement are not stored in the computer's memory;* therefore, away is needed to store data or the results of computations for later use. The LET statement will do this, since *the results* of *computations done in a LET statement are stored in the computer's memory.*

LET Statement

One of the more useful BASIC language keywords is **LET.** It is an instruction to the computer to either assign a specified value to a variable name, or to do certain computations and then assign the result to a variable name. The LET statement is not a statement of algebraic equality; rather, it is a definition that assigns a value or a number to a variable.

The LET statement can be used to assign a constant value to a variable name, a variable to a variable name, or the result of an expression to a variable name. Therefore, the LET statement is often referred to as an *assignment statement.*

For example:

Assigning a constant value to a variable name

```
10 LET A = 212
```

In this example the value 212 is assigned to the variable name A and stored in memory with the location name, A. The equal sign should be read as *be replaced by* or more precisely *be assigned the value of.* This does not represent algebraic equality.

Let's examine a program with a LET statement that assigns a constant value to a variable name. Assume we wanted to find the area of three circles, each with a radius of 5, 6, and 7 inches respectively. We know that pi (3. 1416) will be used in each computation; therefore, if we assign the constant value of 3.1416 to a variable name (P), we can use the variable name in each computation rather than writing 3.1416 three times. This eliminates repetitive coding. The program could be written this way:

Example:

Assigning a constant value to a variable name

```
10   LET P=3.1416
20   PRINT P*(5**2),P*(6**2),P*(7**2)
30   END
RUN
 78.54               113.0976            153.9384
```

If a value is to be used several times in a program, it can be assigned to a variable name and stored in the computer's memory. When that value is to be used, you can reference it by using the variable name. In the previous example that is what we did, we assigned 3.1416 the variable name of P and each time we wanted to use it, we referenced P rather than write 3.1416 each time.

Since an *expression is* really nothing more than a value represented by algebraic symbols, the LET statement can also be used to assign a variable name the value of an expression as in a formula or equation.

Using the previous example of the circles, let's see how this works.

Example:

Assigning the value of an expression to a variable name

```
10   LET P=3.1416
20   LET A1=P*(5**2)
30   LET A2=P*(6**2)
40   LET A3=P*(7**2)
50   PRINT A1,A2,A3
60   END
RUN
 78.54               113.0976            153.9384
```

In this example, the expressions on the right side of the equal sign were calculated and the results assigned to the variable names Al, A2, and A3. Then in line 50 when we wanted to print the answers we referenced the variable names instead of having to recode the expressions.

Three important points to remember about the LET statement are:

- It assigns a value to a variable name.
- The value may be expressed as a constant, another variable or an expression.
- The value is stored in memory and may be referenced by its variable name.

ARITHMETIC EXPRESSIONS

Arithmetic expressions are composed of a combination of constants, variables, operation symbols, and functions. An expression may be very simple or quite complex, but it will result in a single value. Whether an expression is simple or complex, the calculations must be performed in a specific order. To ensure the computer will correctly evaluate and calculate arithmetic expressions, you have to learn to code them using the rules of BASIC. In order to use arithmetic expressions efficiently, you must be able to evaluate and convert conventional mathematical expressions into proper BASIC expressions.

Arithmetic Operators

Unlike algebra, each arithmetic operator in a BASIC expression must be specified by the inclusion of the appropriate operator symbol. The symbols with their operation are as follows:

Operator Symbol	Operation
**	Exponentiation (an up arrow ↑ is used on some computers)
*	Multiplication
/	Division
+	Addition
−	Subtraction

The symbols associated with each operator are standard in the BASIC language. In mathematics it's all right to write AB to mean multiply A times B. In BASIC you must write A*B since default conditions do not exist. This means if you forget one of the operator symbols, the compiler will not insert it for you; but rather will give you an error message.

Precedence Rule

Many expressions are complex and may have two or more operators; therefore, the computer must have specific rules of precedence to define the order of execution. The computer will perform exponentiation first, then multiplication or division, then addition or subtraction. That is, exponentiation has precedence over addition and subtraction. The following list shows the operator symbols, operation, and their precedence of execution.

Operator Symbol	Operation	Precedence
**	Exponentiation	1
*	Multiplication	2
/	Division	2
+	Addition	3
−	Subtraction	3

If two or more operators of the same precedence appear in an expression, then the order of evaluation is from left to right.

Example:

Given: A = 10, B = 6, and C = 7

```
LET X = A + B − C
```

A and B will be added giving 16, then C will be subtracted giving 9.

```
LET X = A − B + C
```

B will be subtracted from A giving 4, then C will be added giving 11.

Parentheses Rule

There are cases where the precedence rule may cause a problem. For example:

$$y = \frac{a}{b + c}$$

The BASIC expression LET Y = A/B+ C would produce undesired results, because A would be divided by B and the result added to C. The solution to this problem is in the use of *parentheses.* If we let parentheses override the order of precedence (but maintain the order of precedence within the parentheses), the result will be satisfactory. Now let's examine the previous example using *parentheses.*

Example:

Using parentheses

$$y = \frac{a}{b+c}$$

```
LET Y=A/(B+C)
```

With the parentheses, B is added to C and the sum of this operation is divided into A, giving the correct result.

Sometimes more than one set of parentheses may be needed to tell the BASIC language in what order to execute the arithmetic operations.

Example:

Parentheses inside parentheses

$$\left(\frac{a+b}{c}\right)^2$$

The BASIC expression to accomplish this would be:

```
LET M=((A+B)/C)**2
```

For this expression to give us the correct results, A and B must be added first, then the sum divided by C, and finally that result is squared. When parentheses within parentheses are used, the innermost parentheses will be evaluated first. Addition has a lower precedence than either division or exponentiation; therefore, A + B must be in the inner parentheses. Division has a lower precedence than exponentiation, so (A+B)/C also must be enclosed in parentheses, ((A+B)/C), to ensure it is performed next.

The important thing to remember is the parentheses maybe used to override the normal order of precedence. The *parentheses rule* says:

- Computations inside parentheses are performed first.
- If there are parentheses inside parentheses, the operations inside the inner pair are performed first.

CONSTANTS AND VARIABLES

Throughout Chapters 2 and 3 we have been using *constants* and *variables* to refer to numeric values. Like everything else in a programming language, there are rules to be learned about the coding and use of both constants and variables.

In the BASIC language, we have two ways to refer to a numeric value: first, by a numeric-constant representing the value, and second, by an arbitrary name, a numeric-variable name, representing the value.

Numeric-constants

A numeric-constant is a decimal number, whose value does not change. It may be a whole number or have a decimal or fractional part. If it has a fractional part it must be expressed as a decimal number such as 3.5 rather than 3 1/2. If the number gets too large, the system software will convert the number to scientific notation. As described in Chapter 2, this is simply a decimal fraction multiplied by a positive or negative power of ten.

Numeric-variable Name

A numeric-variable name is an arbitrary name that you select, and you and the computer system use to refer to some location in memory. The value contained in that location may change or vary during execution of a program.

Example:

C = 5/9(F – 32) is the formula for converting temperature in Fahrenheit to Celsius (centigrade). In this equation 5, 9, and 32 are all constants; that is, they never change. F is the variable name for temperature in Fahrenheit; its value varies as the temperature goes up and down. C is the variable name for the temperature in degrees Celsius; it refers to the location in memory where the solution to the computation is stored, and its value varies as the value of F varies according to the formula.

A numeric-variable name maybe any single letter of the English alphabet (A to Z), or a single letter followed by any single decimal digit (0 through 9). Table 3-1 shows examples of valid and invalid variable names.

Table 3-1.—Numeric-variable Names

Valid	Invalid	Explanation
C	2	must be alphabetic
F2	9K	first character must be alphabetic second character must be numeric
Z5	A83	too many characters, only two characters are allowed
A4	AB	second character must be numeric

String-constants and String-variables

Not all constants and variables are numeric. They may be a series or string of characters such as name ("JOHN DOE") or address ("206 VILLAGE GREEN CIRCLE"). They may be composed of any combination of letters, digits, and special characters and are enclosed in quotation marks. Generally they are called character strings and may be either a string-constant or a string-variable. In BASIC, variable names for string-variables are any single letter (A-Z) followed by a dollar sign ($).

For example:

String-constant

```
PRINT "CENTIGRADE TEMPERATURE"
```

String-variable

```
LET A$ = "206 VILLAGE GREEN CIRCLE"
```

The important things to remember about *constants* and *variables* are:

- Numeric-constants are whole or decimal numbers that do not change throughout a program.
- Variables, referenced by variable names, may represent numeric values or character strings.
- Variables are used when values may change during execution of a program.
- Variable names for numeric values may be a single alphabetic character (A-Z), or one alphabetic character followed by a single numeric digit (0-9).
- Variable names for character strings are a single alphabetic character (A-Z) followed by a dollar sign ($).
- String constants are character strings that do not change throughout a program.

SUMMARY

Simple problems can be solved with BASIC by using only two or three instructions. These are the END, PRINT and LET statements. To use these effectively, you must know how they work and what rules must be followed in using them.

The **END** statement, which must be the last statement in every BASIC program, has two functions. It indicates to the compiler that there are no more BASIC statements for it to translate and it terminates execution of the program.

The **PRINT** statement is used to instruct the computer to output something either on the terminal or the printer. The standard print line in BASIC is divided into print zones or fields of 16 spaces each.

The two punctuation marks used in PRINT statements are the comma and semicolon. A comma used as a separator in a PRINT statement causes standard spacing and a semicolon causes packed spacing.

Information enclosed in quotation marks in a PRINT statement will be printed exactly as it appears in the program.

The **LET** statement can be used to assign a constant value to a variable name, a variable to a variable name, or the results of an expression to a variable name. The equal sign in a LET statement does not indicate algebraic equality, rather it means be assigned the value of. The value assigned by a LET statement is stored in the computer's memory; therefore, it can be referenced by its variable name.

Both the PRINT and LET statements may contain expressions with arithmetic operations. These arithmetic operations must be specified by the appropriate operation symbol. Should you forget to include the symbol, the computer will not insert it for you, but will give you an error message. Arithmetic operations within an expression are executed in a prescribed order of precedence: exponentiation first; multiplication and division next; and addition and subtraction last. Parentheses are used to alter the normal order of precedence. Operations inside parentheses are performed first. If there are parentheses inside parentheses, the operations inside the inner pair are performed first.

Constants and variables are used to refer to numeric values or character strings. A constant is a whole or decimal number or character string whose value does not change. A variable name is an arbitrary name you select and you and the computer use to refer to a value stored in the computer's memory. This value may vary during execution of the program, but can contain only one value at a time.

CHAPTER 3

EXERCISES

1. You have been jogging regularly for six months and you want to know how many miles you have jogged in the six month period. Write a program that will total your miles jogged and print a heading that says:

   MILES JOGGED IN SIX MONTHS ARE

   Your miles for each month are: 182, 178, 174, 170, 187, 183

2. Modify the program from question one to include the computation of average miles jogged per month and a heading that says:

   AVERAGE MILES JOGGED PER MONTH IS

   Leave a blank line between headings.

3. Write a program to convert Fahrenheit temperatures to Celsius and print the results. The Fahrenheit temperature readings are: 76, 88, and 96. The formula C = (F - 32)*5/9.

4. Which of the following are valid numeric-variable names? For those that are not valid, what is wrong with them?

A. R	E. A2
B. R1	F. AA
C. 9	G. M52
D. 9L	H. A$

5. For the following expressions write the BASIC statements to solve the expression.

   A. $a = \frac{x+y}{z}$

   B. $b = \frac{x^2}{y^2}$

   C. $c = \frac{(x+y)^2}{z}$

   D. $d = \left(\frac{x^2}{y+z}\right)^2$

CHAPTER 3

EXERCISE ANSWERS

1.
```
10 LET M = 182 + 178 + 174 + 170 + 187 + 183
20 PRINT "MILES JOGGED IN SIX MONTHS ARE ";M
30 END
RUN
MILES JOGGED IN SIX MONTHS ARE     1074
```

2.
```
10 LET M = 182 + 178 + 174 + 170 + 187 + 183
20 LET A = M/6
30 PRINT "MILES JOGGED IN SIX MONTHS ARE ";M
40 PRINT
50 PRINT "AVERAGE MILES JOGGED PER MONTH IS ";A
60 END
RUN
MILES JOGGED IN SIX MONTHS ARE     1074

AVERAGE MILES JOGGED PER MONTH IS     179
```

3.
```
10 LET F = 76
20 LET F1 = 88
30 LET F2 = 96
40 PRINT (F – 32)*5/9,(F1 – 32)*5/9,(F2 – 32)*5/9
50 END
RUN
 24.44            31.11            35.55
```

4. valid numeric-variable names

   A,B,E

   invalid numeric-variable names

   C. must be alphabetic

   D. first character must be alphabetic

   F. second character must be numeric

   G. variable name can only be two characters

   H. valid only for string-variables

5. A. LET A = (X + Y)/Z

   B. LET B = X**2/Y**2

   C. LET C = (X + Y)**2/Z

   D. LET D = (X**2/(Y + Z))**2